A Detailed Performance Characterization of Columbia using Aeronautics Benchmarks and Applications

Michael Aftosmis, Marsha Berger[*], Rupak Biswas, M. Jahed Djomehri[†], Robert Hood[†], Haoqiang Jin, and Cetin Kiris

NASA Advanced Supercomputing (NAS) Division
NASA Ames Research Center, Moffett Field, CA 94035

Abstract

Columbia is a 10,240-processor supercluster consisting of 20 Altix nodes with 512 processors each, and currently ranked as one of the fastest computers in the world. In this paper, we investigate its suitability as a capability computing platform for aeronautics applications. We present the performance characteristics of Columbia obtained on up to eight computing nodes interconnected via the InfiniBand and/or NUMAlink4 communication fabrics. To perform the assessment, we used a subset of the NAS Parallel Benchmarks, including the multi-zone versions, and three computational fluid dynamics applications of interest to NASA. Our results show that the system holds promise for multinode application scaling to at least 4096 processors.

Keywords: SGI Altix, multi-level parallelism, NAS Parallel Benchmarks, multi-block overset grids, computational fluid dynamics

I. Introduction

During the summer of 2004, NASA acquired and installed Columbia, a 10,240-processor SGI Altix supercluster at its Ames Research Center. It consists of 20 nodes, each of which has 512 Intel Itanium2 processors that share 1TB of main memory using SGI's NUMAlink technology. The 20 nodes are interconnected with InfiniBand[10] as well as 10-gigabit Ethernet. In addition, four of the nodes are connected with NUMAlink4, providing a shared-memory capability computing platform of 2048 CPUs and 4TB of memory. In October 2004, the full Columbia system achieved 51.9 Tflop/s on the Linpack benchmark, passing the Earth Simulator and placing it second on the November 2004 Top500 list.[21]

In this paper, we present an assessment of the suitability of Columbia as a capability computing platform for aeronautics applications. In particular, we investigate computations employing up to 4096 processors, both on the 4-node shared memory portion as well as on InfiniBand-connected subclusters. We also evaluate the available programming approaches for their effect on code performance. To perform the assessment, we used the aeronautics-based NAS Parallel Benchmarks (NPB) and state-of-the-art computational fluid dynamics (CFD) codes, both compressible and incompressible multi-block overset grid Euler/Navier-Stokes applications.[1,7,13]

[*]Professor, Department of Computer Science, Courant Institute, New York University.
[†]Employee of Computer Sciences Corporation.

II. The Columbia Supercluster

Introduced in early 2003, the SGI Altix 3000 systems are an adaptation of the Origin 3000, which use SGI's NUMAflex global shared-memory architecture. Such systems allow access to all data directly and efficiently, without having to move them through I/O or networking bottlenecks. The NUMAflex design enables the processor, memory, I/O, interconnect, graphics, and storage to be packaged into modular components, called "bricks." The primary difference between the Altix and the Origin systems is the C-Brick, used for the processor and memory. This computational building block for the Altix 3700 consists of four Intel Itanium2 processors, 8GB of local memory, and a two-controller ASIC called the Scalable Hub (SHUB). Each C-Brick shares a peak bandwidth of 3.2 GB/s via the NUMAlink interconnection. Each SHUB interfaces to two CPUs, along with memory, I/O devices, and other SHUBs. The Altix cache-coherency protocol is implemented in the SHUB, which integrates both the snooping operations of the Itanium2 and the directory-based scheme used across the NUMAlink interconnection fabric. A load/store cache miss causes the data to be communicated via the SHUB at a cache-line granularity and automatically replicated in the local cache.

The predominant CPU on Columbia is an implementation of the 64-bit Itanium2 architecture, operating at 1.5 GHz, and is capable of issuing two multiply-adds per cycle for a peak performance of 6.0 Gflop/s. The memory hierarchy consists of 128 floating-point registers and three on-chip data caches (32KB L1, 256KB L2, and 6MB L3). The Itanium2 cannot store floating-point data in L1, making register loads and spills a potential source of bottlenecks; however, a relatively large register set helps mitigate this issue. The processor implements the Explicitly Parallel Instruction set Computing (EPIC) technology where instructions are organized into 128-bit VLIW bundles. The Altix 3700 platform uses the NUMAlink3 interconnect, a high-performance custom network with a fat-tree topology that enables the bisection bandwidth to scale linearly with the number of processors.

Columbia is configured as a cluster of 20 SGI Altix nodes (or boxes), each with 512 processors and 1TB of global shared-access memory. Of these 20 nodes, 12 are model 3700 and the remaining eight are model 3700BX2. The BX2 node is essentially a double-density version of the 3700. Each BX2 C-Brick thus contains eight processors, 16GB local memory, and four SHUBs, doubling the processor count in a rack from 32 to 64 and thereby packing more computational power in the same space. The BX2 C-Bricks are interconnected via NUMAlink4, yielding a peak bandwidth of 6.4 GB/s that is twice the bandwidth between bricks on a 3700. In addition, five of the Columbia BX2's use 1.6 GHz (rather than 1.5 GHz) parts and 9MB L3 caches. Table 1 summarizes the main characteristics of the 3700 and BX2 nodes used in Columbia.

Characteristics	3700	BX2 (Type "a")	BX2 (Type "b")
Architecture	NUMAflex, SSI	NUMAflex, SSI	NUMAflex, SSI
# Processors	512	512	512
Packaging	32 CPUs/rack	64 CPUs/rack	64 CPUs/rack
Processor	Itanium2	Itanium2	Itanium2
Clock	1.5 GHz	1.5 GHz	1.6 GHz
L3 cache	6 MB	6 MB	9 MB
Interconnect	NUMAlink3	NUMAlink4	NUMAlink4
Bandwidth	3.2 GB/s	6.4 GB/s	6.4 GB/s
Memory	1 TB	1 TB	1 TB
Th. peak perf.	3.07 Tflop/s	3.07 Tflop/s	3.28 Tflop/s

Table 1. Characteristics of the three types of Altix nodes used in Columbia.

Two communication fabrics connect the 20 Altix systems: an InfiniBand switch[22] provides low-latency MPI communication, and a 10-gigabit Ethernet switch provides user access and I/O communications. InfiniBand is a revolutionary, state-of-the-art technology that defines very high-speed networks for interconnecting compute and I/O nodes.[10] It is an open industry standard for designing high-performance compute clusters of PCs and SMPs. Its high peak bandwidth and comparable minimum latency distinguish it from other competing network technologies such as Quadrics and Myrinet.[14] Four of the 1.6 GHz BX2 nodes are linked with NUMAlink4 technology to allow the global shared-memory constructs to significantly reduce inter-processor communication latency. This 2,048-processor subsystem within Columbia provides a 13 Tflop/s peak capability platform.

A number of programming paradigms are supported on Columbia, including the standard OpenMP and MPI, SGI SHMEM, and Multi-Level Parallelism (MLP). MPI and SHMEM are provided by SGI's Message Passing Toolkit (MPT), while C/C++ and Fortran compilers from Intel support OpenMP. The MLP library was developed by Taft at NASA Ames.[20] Both OpenMP and MLP can take advantage of the globally shared memory within an Altix node. MPI and SHMEM can be used to communicate between Altix nodes connected with the NUMAlink interconnect; however, communication over the InfiniBand switch requires the use of MPI. Because of the hardware limitation on the number of InfiniBand connections through InfiniBand cards installed on each node, the number of per-node MPI processes, k, is confined by

$$k \leq \sqrt{\frac{N_{\mathrm{cards}} \times N_{\mathrm{connections}}}{n-1}}$$

where n (≥ 2) is the number of Altix nodes involved. Currently on Columbia, $N_{\mathrm{cards}} = 8$ per node and $N_{\mathrm{connections}} =$ 64K per card. Thus, a pure MPI code can only fully utilize up to three Altix nodes under the current InfiniBand setup. A hybrid (e.g. MPI+OpenMP) version of applications would be required for runs using four or more nodes.

III. Benchmarks and Applications

A previous paper[6] provides a detailed performance characterization of Columbia's components and the scalability up to 2048 processors. Particular attention was paid to the differences between 3700 and BX2 nodes. Another paper[15] reports the performance of two high-performance aerodynamic simulation packages on up to 2048 CPUs. In this work, we explore the feasibility of using the 2048- and 4096-CPU subclusters of the BX2 nodes for capability computing in aeronautics. We begin by describing the codes and test cases used in our study.

A. NAS Parallel Benchmarks

The NAS Parallel Benchmarks (NPB) are well-known problems for testing the capabilities of parallel computers and parallelization tools. They were derived from computational fluid dynamics (CFD) codes and are widely recognized as a standard indicator of parallel computer performance. The original NPB suite consists of five kernels and three simulated CFD applications, given as "pencil-and-paper" specifications.[3] The kernels mimic the computational core of five numerical methods, while the three simulated applications reproduce much of the data movement and computation found in full CFD codes. Reference implementations were subsequently provided as NPB2,[4] using MPI as the parallel programming paradigm, and later expanded to other programming paradigms (such as OpenMP). Recent effort in NPB development was focused on new benchmarks, including the new multi-zone version, called NPB-MZ.[11] While the original NPB exploits fine-grain parallelism in a single zone, the multi-zone benchmarks stress the need to exploit multiple levels of parallelism for efficiency and to balance the computational load.

For evaluating the Columbia system, we selected a subset of the benchmarks: three kernels (MG, CG, and FT), one simulated application (BT), and two multi-zone benchmarks (BT-MZ and SP-MZ).[4,11] These cover five types of numerical methods found in many scientific applications. Briefly, MG (multi-grid) tests long- and short-distance communication, CG (conjugate gradient) tests irregular memory access and communication, FT (fast Fourier transform) tests all-to-all communication, BT (block-triadiagonal solver) tests nearest neighbor communication, and BT-MZ (uneven sized zones) and SP-MZ (even sized zones) test both coarse- and fine-grain parallelism and load balance. For our experiments, we use both MPI and OpenMP implementations of the four original NPBs and the hybrid MPI+OpenMP implementation of the NPB-MZ from the latest NPB3.1 distribution.[18] A hybrid MPI+MLP version of the NPB-MZ was developed to exploit parallelism at three levels on multinodes of Columbia: MPI across nodes, forked processes within an Altix node, and OpenMP at fine grain. To stress the processors, memory, and network of the Columbia system, we introduced two new classes of problem sizes for the multi-zone benchmarks: Class E (4096 zones, $4224\times3456\times92$ aggregated grid size) and Class F (16384 zones, $12032\times8960\times250$ aggregated grid size).

B. Cart3D: Inviscid Flow Analysis on Adapted Cartesian Meshes

Cart3D is a simulation package targeted at conceptual and preliminary design of aerospace vehicles with complex geometry. It is in widespread use throughout NASA, the DoD, the US intelligence industry, and within dozens of companies in the United States. The flow simulation module solves the Euler equations governing inviscid flow of a compressible fluid. Since these equations neglect the viscous terms present in the full Navier-Stokes equations, boundary-layers, wakes, and other viscous phenomena are not present in the simulations. This simplification removes much of the demand for extremely fine meshing in the wall normal direction that Navier-Stokes solvers must contend with. As a result, the meshes used in inviscid analysis are generally smaller and simpler to generate than those required for viscous solvers. This simplification is largely responsible for both the degree of automation available within the Cart3D package and the speed with which solutions can be obtained. Despite this simplification, inviscid solutions have a large area of applicability within aerospace vehicle design as there are large classes of problems for which they produce excellent results. Moreover, when significant viscous effects are present, large numbers of inviscid solutions can often be corrected using the results of a relatively few full Navier-Stokes simulations.

Cart3D's solver module uses a second-order cell-centered, finite-volume upwind spatial discretization combined with a multigrid-accelerated Runge-Kutta scheme for advance to steady-state.[1] The package automatically adapts the embedded-boundary Cartesian grid to capture control surface deflection of a particular geometry. This flexibility is a key ingredient in automating parameter sweeps over a variety of configurations. Cart3D utilizes several techniques to enhance its efficiency on distributed parallel machines. It uses multigrid for convergence acceleration and employs a domain-decomposition strategy for subdividing the global solution of the governing equations among the processors of a parallel machine.[1,2,5]

To assess performance of Cart3D's solver module on realistically complex problems, several performance experiments were devised examining scalability for a typical large grid case. The case considered here is based on a full Space Shuttle Launch Vehicle (SSLV) example. The mesh (shown in Fig. 1) contains approximately 4.7M cells with 14 levels of adaptive subdivision.

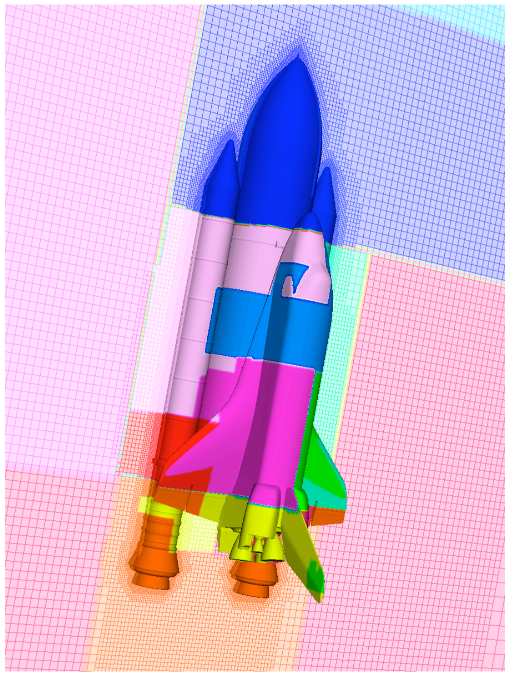

Figure 1. Cartesian mesh around full SSLV configuration including orbiter, external tank, solid rocket boosters, and fore and aft attach hardware. Mesh color indicates 16-way decomposition of 4.7M cells using the SFC partitioner.[2]

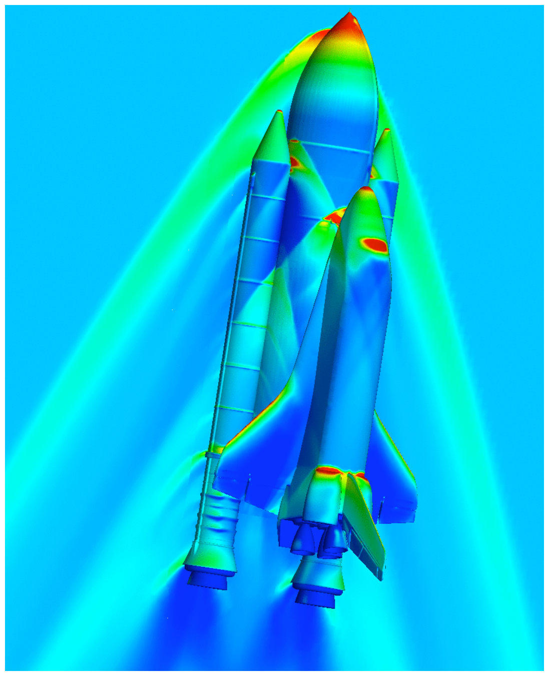

Figure 2. Pressure contours around full SSLV configuration including orbiter, external tank, solid rocket boosters, and fore and aft attach hardware for the benchmarking case described in the text.

The mesh is illustrated with a single cutting plane through the domain. The grid is painted to indicate its partitioning into 16 subdomains using the Peano-Hilbert SFC. Partition boundaries in this example were chosen for perfect load-balancing on homogeneous CPU sets and cut-cells were weighted 2.1 times more heavily than uncut Cartesian hexahedra. The partitions are all predominantly rectangular, which is characteristic of subdomains generated with SFC-based partitioners. Details of the partitioning strategy used in the Cart3D solver module is described elsewhere.[1,2,5] The communication wrapper routines perform explicit data exchanges using direct shared-memory structure copies when linked with the OpenMP library, and use explicitly packed communication buffers when built with MPI. Memory requirements for the solver module are approximately 240 words/cell when using 4-level multigrid. Without multigrid, these requirements drop to about 180 words/cell.

For scalability testing, the mesh density was increased to 25M cells, which is about twice as fine as that shown in Fig. 1. With the storage requirements outlined above, this example consumes a total of about 22GB of memory. Cart3D's solver module solves five equations for each cell in the domain giving this example approximately 125M degrees-of-freedom. The geometry includes detailed models of the orbiter, solid rocket boosters, external tank, five engines, and all attach hardware. The geometry also includes modifications made to the external tank geometry as part of NASA's Return-to-Flight effort. Figure 2 shows pressure contours of the discrete solution at 2.6 Mach, 2.09 degrees angle-of-attack, and 0.8 degrees sideslip. The surface triangulation contains about 1.7M elements. An aerodynamic performance database and virtual-flight trajectories using this configuration with power on was presented in 2004.[17]

This example was used for several performance experiments on the Columbia system, including comparisons between OpenMP and MPI, the effects of multigrid on scalability, and comparisons of the NUMAlink and InfiniBand communication fabrics. Unless otherwise stated, all results presented in this paper used four levels of multigrid.

C. INS3D: Turbopump Flow Simulations

Computations for unsteady flow through a full-scale low-pressure rocket pump are performed utilizing the INS3D computer code.[12] Liquid rocket turbopumps operate under severe conditions and at very high rotational speeds. The low-pressure-fuel turbopump creates transient flow features such as reverse flows, tip clearance effects, secondary flows, vortex shedding, junction flows, and cavitation effects. Flow unsteadiness originating from the inducer is considered to be one of the major contributors to the high frequency cyclic loading that results in cycle fatigue. The reverse flow originating at the tip of an inducer blade travels upstream and interacts with the bellows cavity. To resolve the complex geometry in relative motion, an overset grid approach is employed where the problem domain is decomposed into a number of simple grid components.[7] Connectivity between neighboring grids is established by interpolation at the grid outer boundaries. Addition of new components to the system and simulation of arbitrary relative motion between multiple bodies are achieved by establishing new connectivity without disturbing the existing grids.

The computational grid used for the experiments reported in this paper consisted of 66 million grid points and 267 blocks (or zones). Details of the grid system are shown in Fig. 3. Figure 4 displays particle traces colored by axial velocity entering the low-pressure fuel pump. The blue particles represent regions of positive axial velocity, while the red particles indicate four back flow regions. The gray particles identify the stagnation regions in the flow.

The INS3D code solves the incompressible Navier-Stokes equations for both steady-state and unsteady flows. The numerical solution requires special attention in order to satisfy the divergence-free constraint on the velocity field. The incompressible formulation does not explicitly yield the pressure field from an equation of state or the continuity

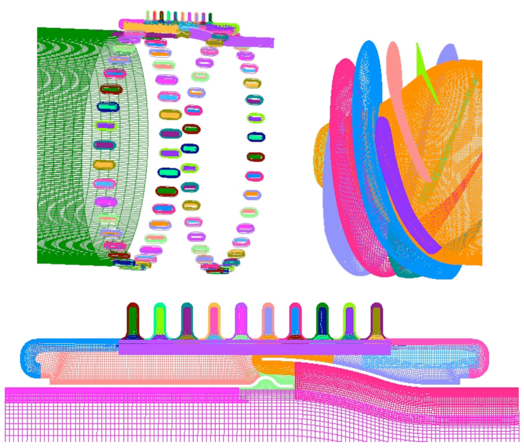
Figure 3. Surface grids for the low pressure fuel pump inducer and the flowliner.

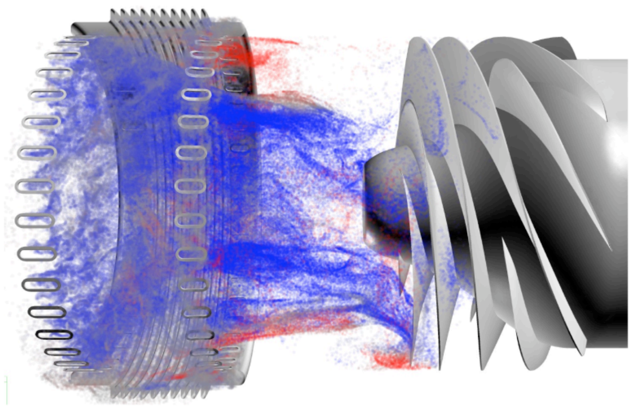
Figure 4. Instantaneous snapshot of particle traces colored by axial velocity values.

equation. One way to avoid the difficulty of the elliptic nature of the equations is to use an artificial compressibility method that introduces a time-derivative of the pressure term into the continuity equation. This transforms the elliptic-parabolic partial differential equations into the hyperbolic-parabolic type. To obtain time-accurate solutions, the equations are iterated to convergence in pseudo-time for each physical time step until the divergence of the velocity field has been reduced below a specified tolerance value. The total number of sub-iterations required varies depending on the problem, time step size, and the artificial compressibility parameter. Typically, the number ranges from 10 to 30 sub-iterations. The matrix equation is solved iteratively by using a non-factored Gauss-Seidel type line-relaxation scheme, which maintains stability and allows a large pseudo-time step to be taken. More detailed information about the application can be found elsewhere.[12, 13]

Single-node performance results reported in this paper were obtained for computations carried out using the Multi-Level Parallelism (MLP) paradigm for shared-memory systems.[20] All data communications at the coarsest and finest levels are accomplished via direct memory referencing instructions. The coarsest level parallelism is supplied by spawning off independent processes via the standard UNIX fork. A library of routines is used to initiate forks, to establish shared memory arenas, and to provide synchronization primitives. The boundary data for the overset grid system is archived in the shared memory arena by each process. Fine grain parallelism is obtained by using OpenMP compiler directives. In order to run a 66 million grid point case, the code requires 100GB of memory and approximately 80 microseconds per grid point per iteration.

Performance results on multiple Altix nodes were obtained using the hybrid MPI+OpenMP version of INS3D. The hybrid code uses an MPI interface for coarse-grain parallelism, and OpenMP directives for fine-grain parallelism. Implementation of the parallel strategy starts by assembling the grid zones into groups, each of which is mapped onto an MPI process. Master-worker and point-to-point protocols have been implemented for MPI communications. In the master-worker paradigm, when overlapping grid communication is performed, each group sends its information to a master group. Once the master group has received and processed all of the information, the data is sent to the other groups and computation proceeds. In the point-to-point method, overlapping grid boundary information is updated via MPI asynchronous calls. Performance results obtained by using both communication protocols are reported in the multinode results section.

D. OVERFLOW-D: Rotor Vortex Simulations

For solving the compressible Navier-Stokes equations, we selected the NASA production code called OVERFLOW-D.[16] The code uses the same overset grid methodology[7] as INS3D to perform high-fidelity viscous simulations around realistic aerospace configurations. OVERFLOW-D is popular within the aerodynamics community due to its ability to handle complex designs with multiple geometric components. It is explicitly designed to simplify the modeling of problems when components are in relative motion. The main computational logic at the top level of the sequential code consists of a time-loop and a nested grid-loop. Within the grid-loop, solutions to the flow equations are obtained on the individual grids with imposed boundary conditions. Overlapping boundary points or inter-grid data are updated from the previous time step using an overset grid interpolation procedure. Upon completion of the grid-loop, the solution is automatically advanced to the next time step by the time-loop. The code uses finite difference schemes in space, with a variety of implicit/explicit time stepping.

The hybrid MPI+OpenMP version of OVERFLOW-D takes advantage of the overset grid system, which offers a natural coarse-grain parallelism.[8] A bin-packing algorithm clusters individual grids into groups, each of which is then assigned to an MPI process. The grouping strategy uses a connectivity test that inspects for an overlap between a pair of grids before assigning them to the same group, regardless of the size of the boundary data or their connectivity to other grids. The grid-loop in the parallel implementation is subdivided into two procedures: a group-loop over groups, and a grid-loop over the grids within each group. Since each MPI process is assigned to only one group, the group-loop is executed in parallel, with each group performing its own sequential grid-loop. The inter-grid boundary updates within each group are performed as in the serial case. Inter-group boundary exchanges are achieved via MPI asynchronous communication calls. OpenMP parallelism is achieved by explicit compiler directives inserted at the loop level. The logic is the same as in the pure MPI case, only the computationally intensive portion of the code (i.e. the grid-loop) is multi-threaded via OpenMP.

OVERFLOW-D was originally designed to exploit vector machines. Because Columbia is a cache-based superscalar architecture, modifications were necessary to improve performance. The linear solver of the application, called LU-SGS, was re-implemented using a pipeline algorithm[8] to enhance efficiency which is dictated by the type of data dependencies inherent in the solution algorithm.

Our experiments reported here involve a Navier-Stokes simulation of vortex dynamics in the complex wake flow region around hovering rotors. The grid system consisted of 1679 blocks of various sizes, and approximately 75 million grid points. Figure 5 shows a sectional view of the test application's overset grid system (slice through the off-body wake grids surrounding the hub and rotors) while Fig. 6 shows a cut plane through the computed wake system including vortex sheets as well as a number of individual tip vortices. A complete description of the underlying physics and the numerical simulations pertinent to this test problem can be found elsewhere.[19]

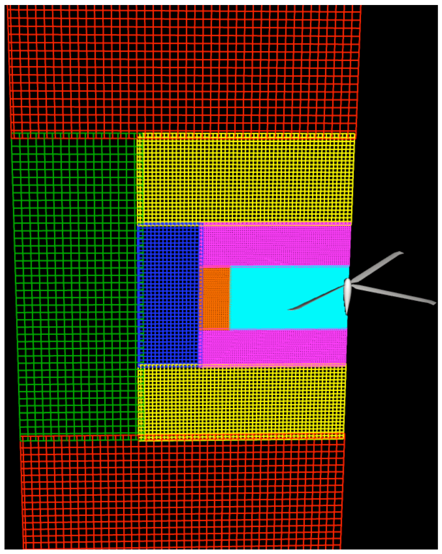
Figure 5. A sectional view of the overset grid system.

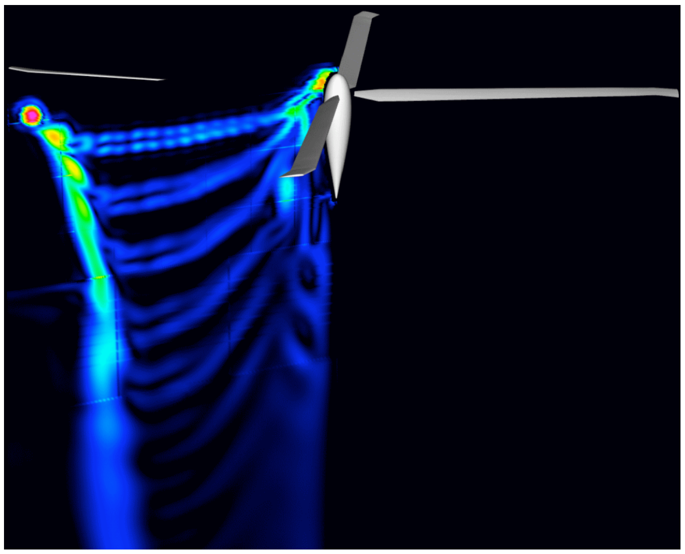
Figure 6. Computed vorticity magnitude contours on a cutting plane located 45° behind the rotor blade.

The memory requirement for OVERFLOW-D is about 40 words per grid point; thus approximately 22GB are necessary to run the test problem used in this study. Note that this requirement gradually increases with the number of processors because of grid and solution management overhead. Due to the overset grid structure, disparate sizes of grid blocks, and grouping strategy for load balancing, no nearest neighbor techniques can be employed. The MPI communication pattern is all-to-all, i.e. each MPI process communicates with all other processes. The total number of send/receive buffers exchanged is of the order $O(n^2)$, where n is the number of MPI tasks. The communication time is typically 20% of the execution time, but could vary significantly with the physics of the problem, its domain and topology, the nature of overlapping blocks, and the number of processors used.

IV. Performance Results

We conducted several experiments using benchmarks and full-scale applications to obtain a detailed performance characterization of Columbia. Results of these experiments are presented in the following subsections.

A. Single-box Results

Our study started with a characterization of the three types of Altix nodes that make up Columbia—the 3700 and the two types of BX2. As a shorthand notation, we will call the BX2 with 1.5 GHz CPUs and 6MB caches a "BX2a", while the BX2 with faster clock and larger cache is denoted "BX2b". This part of the study also investigated the impact of programming paradigm choice on performance.

1. NAS Parallel Benchmarks

Figure 7 shows the per-processor Gflop/s rates reported from runs of both MPI and OpenMP versions of CG, FT, MG, and BT benchmarks on three types of the Columbia nodes, a horizontal line indicating linear scaling. MPI versions of

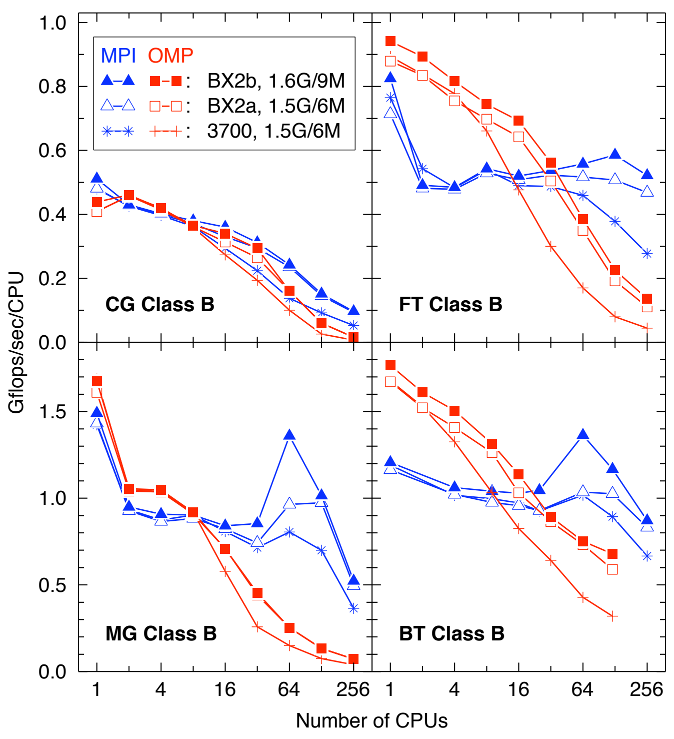

Figure 7. NPB performance comparison on three types of the Columbia nodes.

the benchmarks employ a parallelization strategy of domain decomposition in multiple dimensions to distribute data locally onto each processor, while OpenMP versions simply exploit loop-level parallelism in a shared-address space. These approaches are representative of real world applications where a serial program is parallelized using either MPI or OpenMP.

The effect of doubled network bandwidth and shorter latency of BX2 as compared to 3700 on OpenMP performance is evident: the four OpenMP benchmarks scaled much better on both types of BX2 than on 3700 when the number of threads is four or more. With 128 threads, the difference can be as large as 2x for both FT and BT. The bandwidth effect on MPI performance is less profound until a larger number of processes ($\geq$32) when communication starts to dominate. Observe that on 256 processors, FT runs about twice as fast on BX2 than on 3700, indicating the importance of bandwidth for the all-to-all communication used in the benchmark.

A bigger cache (9MB) in the BX2b node produced substantial performance improvement for the MPI codes for large number of processors (e.g. the peaks at 64 CPUs for MG and BT) when the data can fit into local cache on each processor. On the other hand, no significant difference for the OpenMP codes is observed, primarily because the cost of accessing shared data from each OpenMP thread increases substantially as the number of CPUs increases, which overwhelms any benefit from a larger cache size. In the case of MPI, the falloff from the peak is due to the increased communication-to-computation ratio (a fixed problem size implies data per processor is decreasing as the number of processors increases) as occurred earlier in the OpenMP codes. The slightly larger processor speed of BX2b (1.6 GHz) brings only marginal performance gain, as illustrated from the OpenMP FT and BT results.

Although OpenMP versions of NPB demonstrated better performance on a small number of CPUs, accessing local data and carefully managing communications in the MPI codes produced significantly better scaling than the OpenMP codes that use a simple loop parallelization strategy and cannot be easily optimized for accessing shared data.

2. *Cart3D*

The domain-decomposition parallelization strategy in the Cart3D flow simulation package has previously demonstrated excellent scalability on large numbers of processors with both MPI and OpenMP communication libraries.[5]

This behavior makes it a good candidate for comparison of the performance of BX2a and BX2b nodes using large numbers of processors on a complete application. Figure 8 shows stacked charts of parallel speedup and execution timings for Cart3D using both MPI and OpenMP. Line graphs in the figure show parallel speedup on 32–474 CPUs; the corresponding execution time for five multigrid cycles is shown via bar charts. The scalability data assume perfect speedup on 32 processors which was the smallest CPU set run.

The parallel speedup results show nearly ideal performance for OpenMP and MPI builds on both BX2a and BX2b nodes. The best among these is the MPI-build on the BX2b node which achieves a parallel speedup of 473.8 on 474 CPUs. The OpenMP version on the same node is nearly as good with a parallel speedup of about 472. Interestingly, the scalability data on BX2a is slightly worse, with both OpenMP and MPI builds showing parallel speedups of around 380 on 474 CPUs. This result is counter-intuitive. Both BX2a and BX2b nodes have the same routers while the BX2b CPUs have slightly faster clocks and bigger caches. Single CPU timing tests with Cart3D's solver module show 5-6 percent higher throughput with the BX2b's CPUs. As a result of this higher compute speed with essentially the same interconnect, intuition predicts better scalability on the BX2a's, (and better raw timings on the BX2b's). Close examination of the parallel speedup data is required to resolve this apparent disparity between intuition and the actual performance data. Despite its higher data consumption rates, scalability is very nearly perfect on the BX2b system. This implies that with its lower data consumption rates and identical interconnect, the code should be able to achieve speedup results at least as good on the BX2a system. The fact that it is showing slightly worse performance is most likely due to a load imbalance on this system. Both systems used identical decompositions of the computational meshes in the various simulations. However, in Cart3D, good load balancing needs build-time weighting of the work required for the two primary cell types found in the cut-cell Cartesian meshes. These weights are typically tuned using numerical experiments and are fixed for a given platform. The executable used to generate the data in Fig. 8 was tuned using a BX2b system, resulting in a slight load imbalance on large numbers of CPUs on the BX2a. This hypothesis is supported by timings collected for both the MPI and OpenMP runs that show an increasing performance advantage for BX2b over BX2a with larger CPU counts. Since the BX2b system is showing essentially ideal scalability, the load imbalance shows up as decreasing parallel efficiency on the BX2a, making the BX2b look artificially good. Our final observation for this application is that, even with the higher data consumption rates on the BX2b systems, the BX2 routers provide sufficient communication bandwidth to support the CPUs without adverse impact on a single Columbia node.

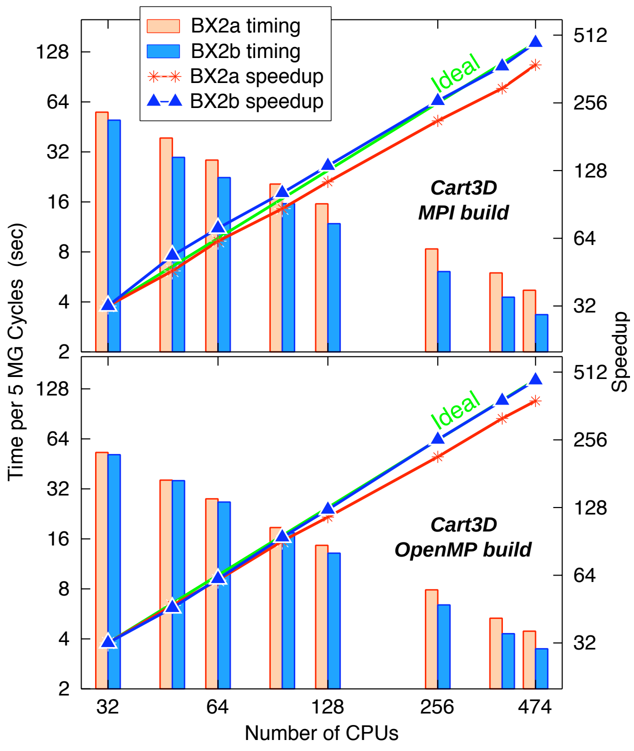

Figure 8. Comparison of execution time and parallel speedup of Cart3D solver module on single BX2a and BX2b nodes of Columbia system using both MPI and OpenMP communication libraries.

3. *INS3D*

Two distinct parallel processing paradigms are used in the INS3D code—the Multi-Level Parallel (MLP) and the MPI+OpenMP hybrid parallel programming models. Both contain coarse- and fine-grain parallelism. Coarse-grain parallelism is achieved through a UNIX fork in MLP and through explicit message passing in the MPI+OpenMP code. Fine-grain parallelism is achieved using OpenMP compiler directives in both implementations. Both codes use a group-based data structure for global solution arrays. The MLP version uses a global shared memory data structure for overset connectivity arrays, while the MPI+OpenMP code uses local copies of the connectivity arrays providing a more local data structure. Computations were performed to compare the scalability between the MLP and

MPI+OpenMP hybrid (using point-to-point communication protocol) versions of the INS3D code on the Columbia system using the BX2b processors. Initial computations using one group and one thread were used to establish the baseline runtime for one physical time step, where 720 such time steps are required to complete one inducer rotation.

Figure 9 displays the time per iteration (in minutes) versus the number of CPUs and the speedup factor for both codes. Here, 36 groups have been chosen to maintain good load balance for both versions. Then the runtime per physical time step is obtained using various numbers of OpenMP threads (1, 2, 4, 8, and 14). It includes the I/O time required to write the time-accurate solution to disk at each time step.

The scalability for a fixed number of both MLP and MPI groups and varying OpenMP threads is good, but begins to decay as the number of OpenMP threads becomes large. Further scaling can be accomplished by fixing the number of OpenMP threads and increasing the number of MLP/MPI groups until the load balancing begins to fail. Unlike varying the OpenMP threads which does not affect the convergence rate of INS3D, varying the number of groups may deteriorate the rate. This will lead to more iterations even though faster runtime per iteration is achieved. Figure 9 shows that the MLP and MPI+OpenMP codes perform almost equivalently for one OpenMP thread, but then as the number of threads are increased, the MPI+OpenMP version begins to perform slightly better than the MLP implementation. This can be attributed to having local copies of the connectivity arrays in the MPI+OpenMP hybrid code. Having the MPI+OpenMP version of INS3D as scalable as the MLP code is promising since this hybrid version is easily portable to other platforms.

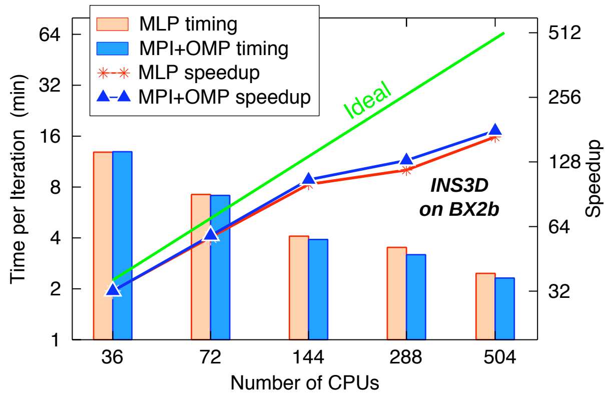

Figure 9. INS3D performance on BX2b: programming paradigm comparison.

4. *Overflow-D*

The performance of OVERFLOW-D was also evaluated on Columbia using the 3700 and BX2b processors. Figure 10 shows total execution times of the application per time step. A typical production run requires about 50,000 such time steps. For various number of processors, we report timings from the best combination of processes and threads.

Observe that execution time on BX2b is significantly smaller compared to 3700 (e.g. more than a factor of 3x on 508 CPUs). On average, OVERFLOW-D runs almost 2x faster on the BX2b than the 3700. In addition, the communication time (not shown) is also reduced by more than 50%.

The performance scalability on the 3700 is reasonably good up to 64 processors, but flattens beyond 256. This is due to the small ratio of grid blocks to the number of MPI tasks that makes balancing computational workload extremely challenging. With 508 MPI processes and only 1679 blocks, it is difficult for any grouping strategy to achieve a proper load balance. Various load balancing strategies for overset grids are extensively discussed elsewhere.[9]

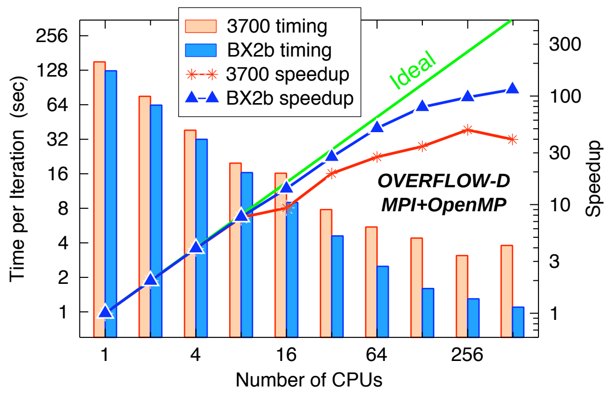

Figure 10. OVERFLOW-D performance on 3700 and BX2b.

Another reason for poor 3700 scalability on large processor counts is insufficient computational work per processor. This could be verified by examining the ratio of communication to execution time. This ratio is about 0.3 for 256 processors, but increases to more than 0.5 on 508 CPUs. For our problem consisting of 75 million grid points, there are only about 150,000 grid points per MPI task, which is too little for Columbia's fast processors compared to the communication overhead. The problem used here was initially built for production runs on platforms having fewer processors with smaller caches and slower clock rates.

Scalability on the BX2b is significantly better. For example, OVERFLOW-D efficiency for 128, 256, and 508 processors is 61%, 37%, and 27% (compared to 26%, 19%, and 7% on the 3700). In spite of the same load imbalance problem, the enhanced bandwidth on the BX2b significantly reduces the communication times. The increased bandwidth is particularly important at the coarse-grain level of OVERFLOW-D, which has an all-to-all communication pattern every time step. This is consistent with our experiments conducted on the NPBs and reported in Section IV.A.1. The reduction in the BX2b computation time can be attributed to its larger L3 cache and maybe its faster CPU speed.

B. Multinode Results

We next reran a subset of our experiments on the 2048-processor BX2b subsystem that is connected with both NUMAlink4 and InfiniBand switches. We also performed tests on up to eight BX2 Altix nodes. These results are presented in the following subsections.

1. NAS Parallel Benchmarks

The hybrid MPI+OpenMP codes of BT-MZ and SP-MZ were tested across four Columbia nodes connected with both the NUMAlink4 network and the InfiniBand switch. We used the Class E problem (4096 zones, 1.3 billion aggregated grid points) for these tests. The top row of Fig. 11 compares the per-CPU Gflop/s rates obtained from runs using NUMAlink4 with those from within a single Altix BX2b node. The two sets of data represent runs with one and two OpenMP threads per MPI process, respectively. For 512 CPUs or less, the NUMAlink4 results are comparable to or even better than the in-node results. In particular, the performance of 512-processor runs in a single node dropped by 10–15%, primarily because these runs also used the CPUs that were allocated for systems software (called *boot cpuset*), which interfered with our tests. Reducing the number of CPUs to 508 improves the BT-MZ performance within a node.

Since MPI is used for coarse-grain parallelism among zones for the hybrid implementations, load balancing for SP-MZ is trivial as long as the number of zones is divisible by the number of MPI processes. The uneven-size zones in BT-MZ allows more flexible choice of the number of MPI processes; however, as the number of CPUs increases, OpenMP threads may be required to get better load balance (and therefore better performance). This is evident from the BT-MZ results in Fig. 11. There is about 11% performance improvement from runs using two OpenMP threads versus

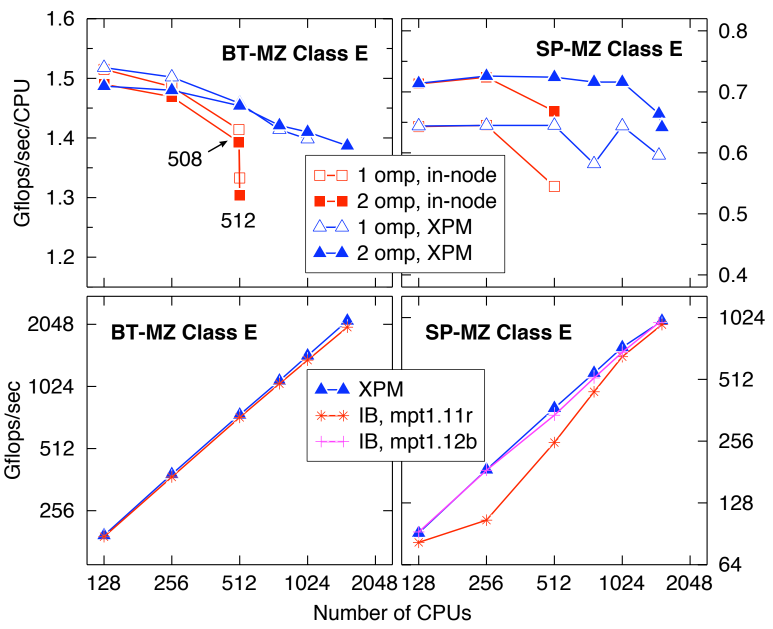

Figure 11. Comparison of NPB-MZ performance under three different networks: in-node, NUMAlink4 (XPM), and InfiniBand (IB).

one (e.g. 256×2 vs. 512×1) for the SP-MZ benchmark. This effect could be attributed to less MPI communication when two threads are used. The performance drop for SP-MZ at 768 and 1536 processors can be explained by load imbalance for these CPU counts.

The bottom row of Fig. 11 compares the total Gflop/s rates from runs using NUMAlink4 to those using InfiniBand, taking the best process-thread combinations. Note a close-to-linear speedup for BT-MZ. The InfiniBand results are only about 7% worse. On the other hand, we noticed anomalous InfiniBand performance for SP-MZ when a released SGI MPT runtime library (mpt1.11r) was used. In fact, on 256 processors, the InfiniBand result is 40% slower than NUMAlink4, but the InfiniBand performance improves as the number of CPUs increases. We used a beta version of the MPT library (mpt1.12b) and reran some of the data points. As shown in the lower right of Fig. 11, the beta version of the library produced InfiniBand results that are very close in performance to the NUMAlink4 results. As it turned out, the InfiniBand MPI performance is sensitive to the settings for a few SGI MPT parameters that control how MPI accesses its internal message buffers. Specifically, we had to increase `MPI_BUFS_PER_HOST` and `MPI_BUFS_PER_PROC` by a factor of eight from the default values in order to obtain the good performance.

As discussed in Section II, running applications across multiple Altix nodes on Columbia usually requires a hybrid approach because of the limitation of the InfiniBand switch imposed on the number of MPI processes allowed on each node. To work around this limitation, one can increase the number of OpenMP threads on each MPI process. However, increasing the number of OpenMP threads does not necessarily produce the desired parallel efficiency.[6] In our tests, we have adopted a three-level approach, MPI+MLP: using MPI and forked processes for coarse grained parallelism, and OpenMP for fine grained parallelism. The forked MLP processes communicate through shared memory buffers. The three-level approach is a natural fit to the Columbia architecture.

A 4032-processor run of the BT-MZ Class E problem was conducted on eight BX2 nodes, five of them being BX2b and three BX2a. We used a total of 504 MPI processes, each MPI process then forked one additional process, and each process used four OpenMP threads (as indicated by a notation "504×2×4"). Figure 12a shows computation

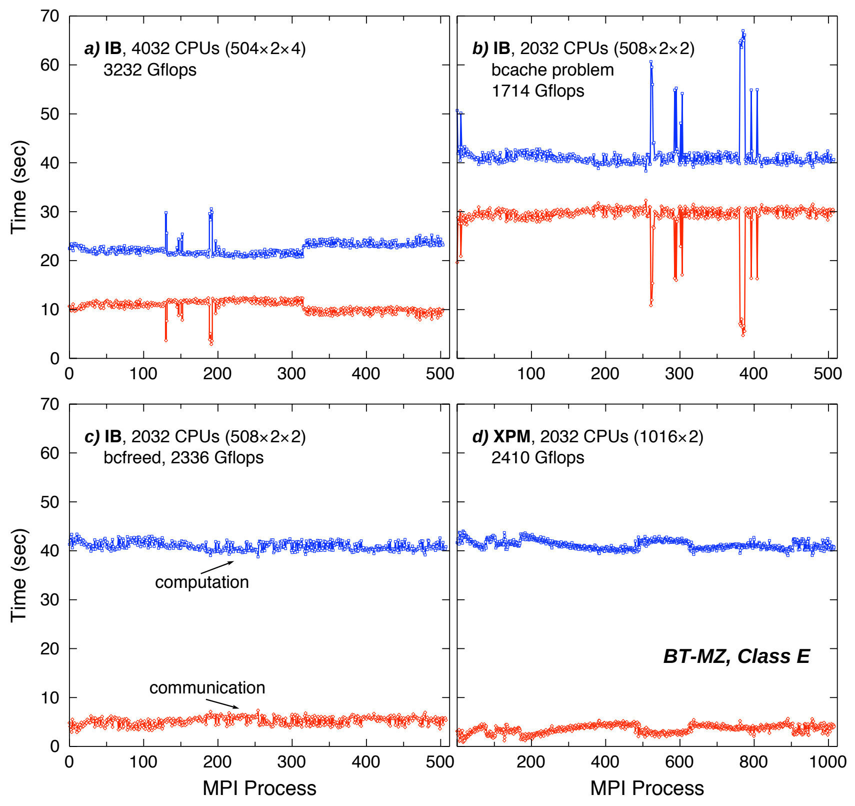

Figure 12. Timing profiles of the BT-MZ Class E problem from multi-box runs: a) 4032 CPUs with InfiniBand (IB), b) 2032 CPUs with IB, having a buffer cache problem, c) 2032 CPUs with IB and the buffer cache problem fixed, and d) 2032 CPUs with NUMAlink4 (XPM). The quoted Gflop/s numbers are reported by the benchmark.

and communication time profiles reported by each MPI process from this run. The communication time includes barrier synchronizations, thus, a near mirror reflection of the computation time is observed in the plot. An increase in computation time after MPI rank 315 is due to the processor difference between BX2b (1.6 GHz) and BX2a (1.5 GHz) that the benchmark implementation did not take into account in its load balance algorithm. Clearly there are a few unexpected spikes in computation time at distinct MPI ranks (130, 147, 152, and 190–193). These spikes have caused substantial increase in computation time (as much as 50%) at these points and in communication time at all other MPI processes. Another run with 2032 CPUs (see Fig. 12*b*) using four BX2b nodes and IB connection showed a similar behavior: spikes in computation time appeared at MPI ranks 261, 295, 303, and 381–387, which were correlated to those in the 4032-CPU run. Further investigations have shown that these spikes correspond to the same set of CPUs in one of the BX2b boxes and the local memory associated with these CPUs were totally filled by system buffer caches. This has caused the allocation of remote memory to user jobs when these CPUs were used. Evidently the use of remote memory significantly increased computation time in the BT-MZ run. After correcting the problem by freeing system buffer caches, we reran the 2032-CPU case and did not observe any abnormal spikes (see Fig. 12*c*). The performance also improved from 1714 Gflop/s to 2336 Gflop/s (36% improvement). An earlier 2032-CPU run using the NUMAlink4 connection did not have a similar problem (see Fig. 12*d*. Note: the run used 1016 MPI processes and two OpenMP threads per process). Unfortunately we did not have chance to confirm the improvement for the 4032-CPU run, which we would expect to be about 26% (or from 3232 Gflop/s to 4072 Gflop/s) when load imbalance is considered.

2. *Cart3D*

As discussed earlier, Cart3D's solver module can be built against either OpenMP or MPI communication libraries. On the Columbia system, the memory on each 512-CPU node is globally sharable by any process within the node, but cache coherency is not maintained between nodes. Thus, pure OpenMP codes are restricted to, at most, the 512 CPUs within a single box. As a result of this restriction, all the multinode examples with Cart3D are run using the MPI communication back-end, and the numerical experiments focus on the effects of increasing the number of multigrid levels in the solution algorithm, and comparing the performance of the NUMAlink and Infiniband connection fabrics. These experiments were carried out on four BX2b Columbia nodes.

Figure 13 examines parallel speedup for the system comparing the baseline four-level multigrid solution algorithm with single grid. This experiment was carried out exclusively using the NUMAlink interconnect, and spanned up to 2016 CPUs on 1–4 BX2b nodes. Reducing the number of multigrid levels in the solution algorithm reduces the inter-subdomain communication much faster than it decreases computation. The net result is that this mode of operation de-emphasizes communication (relative to floating-point performance) in the solution algorithm. Scalability for the single grid scheme is very nearly ideal, achieving parallel speedups of about 1900 on 2016 CPUs. By contrast, the figure shows that even on the NUMAlink, communication performance is beginning to affect scalability with four levels of multigrid. This is not surprising: with only 25M cells in the fine mesh (12,000 cells/partition on 2016 CPUs), the coarsest mesh in the multigrid sequence has only 32,000 cells giving only a scant 16 cells per partition on 2016 CPUs. A slight falloff in the multigrid results starts appearing around 688 CPUs, but does not really start to degrade until above 1024 CPUs. Given this relatively modest decrease in performance, it seems clear that the bandwidth demands of the solver are not greatly in excess of that delivered by the NUMAlink. With 2016 CPUs and four levels of multigrid, the NUMAlink still posts parallel speedups of about 1585.

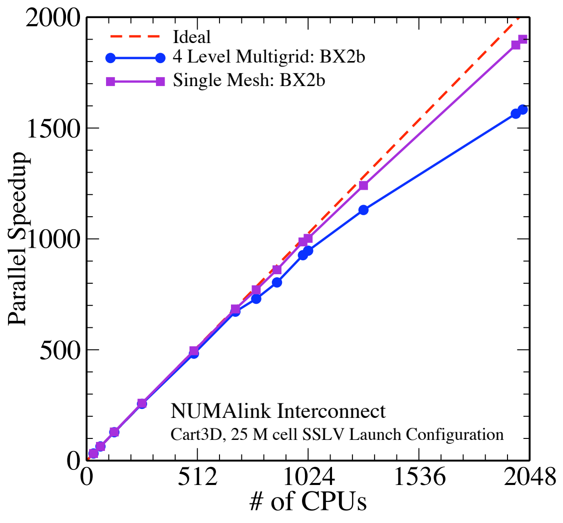

Figure 13. Parallel speedup of Cart3D solver module using one and four levels of mesh in the multigrid hierarchy with NUMAlink interconnect.

Our previous work[6] included a study of delivered bandwidth and latency for both NUMAlink and InfiniBand for a variety of different communication patterns. To understand the implications of this study for Cart3D's solver module, the baseline four-level multigrid scheme was re-run using the InfiniBand interconnect on the same BX2b nodes as the preceding experiment. Figure 14 displays these results plotted against those of the NUMAlink interconnect.

As before, the identical problem was run on 32 to 2016 CPUs using MPI. Note that results with InfiniBand, however, do not extend beyond 1524 CPUs due to a limitation on the number of InfiniBand connections. Results show that between 32 and 496 CPUs, the cases were run on a single node and thus there is no difference between the two curves (no node-to-node communication). Cases with 508–1000 CPUs were run spanning two nodes of Columbia and some interesting differences begin to appear. While the InfiniBand performance consistently lags that of the NUMAlink, the most striking example is at 508 CPUs which actually underperforms the single-node case with 496 CPUs. This is consistent with previous observations[6] which quantify the decrease in delivered bandwidth for InfiniBand across two nodes. This work also predicts an increasing penalty when spanning four nodes. As expected, cases with 1024–2016 CPUs (run on four nodes) show a further decrease with respect to those posted by the NUMAlink.

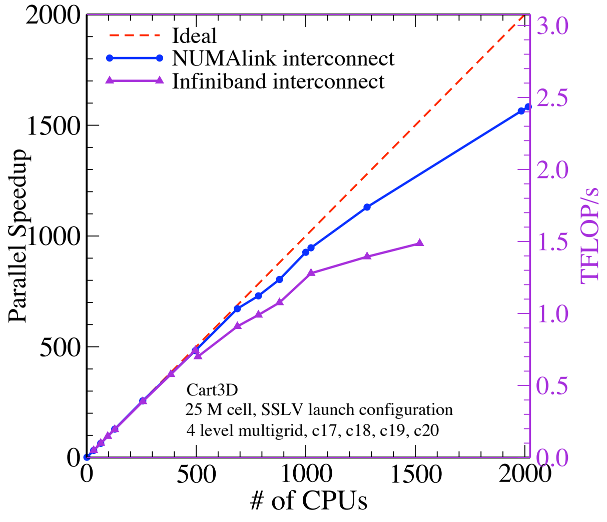

Figure 14. Comparison of parallel speedup and Tflop/s of Cart3D solver module with four levels of multigrid using NUMAlink and InfiniBand.

The right axis of the speedup plot in Fig. 14 is scaled in Tflop/s for the baseline solution algorithm. The number of floating-point operations in these experiments were obtained by interrogating the Itanium2's hardware counters using Intel's "pfmon" interface. Operations were counted for a single multigrid cycle and then divided by the time per iteration on various numbers of processors to provide this scale (a single MADD was counted as two operations). Substantial work on optimizing single CPU performance with this code has resulted in somewhat better than 1.5 Gflop/s on each CPU. When combined with linear parallel speedup, this produces about 0.75 Tflop/s for the code on 496 processors of a single Columbia node. Performance of the NUMAlink case with 2016 CPUs is slightly over 2.4 Tflop/s.

3. *INS3D*

We compare the performance of the INS3D MPI+OpenMP code on multiple BX2b nodes against single node results. This includes running the MPI+OpenMP version using two different communication paradigms: master-worker and point-to-point. The runtime per physical time step is recorded using 36 MPI groups and 1, 4, 8, and 14 OpenMP threads on one, two, and four BX2b nodes. Communication between nodes is achieved using the InfiniBand and NUMAlink4 interconnects, denoted as IB and XPM respectively.

Figure 15 contains results using the point-to-point communication paradigm. When comparing the performance of using multiple nodes with that of a single node, we observe that the scalability of the multinode runs with NUMAlink4 is similar to the single node runs (which also use NUMAlink4 internally). However, when using InfiniBand, the execution time per iteration increases by 10–29% on two and four node runs. The difference between the two- and four-node runs decreases as the number of CPUs increases.

Figure 16 displays the results using the master-worker communication paradigm. Note that the time per iteration is much higher using this protocol compared to the point-to-point communication. We also see a significant deterioration in scalability for both single and multinode runs. With NUMAlink4, we observe a 5–10% increase in runtime per iteration from one to two nodes and an 8–16% increase using four nodes. This is because the master resides on one node and all workers on the other nodes must communicate with the master. Alternatively, when using point-to-point communication, many of the messages remain within the node from which they are sent. In fact, the MPI groups can be manipulated so that a very small number of messages (as low as one in many cases) must be passed between each node. Note that this optimization has not been utilized here and will be studied further. An additional 14–27% increase

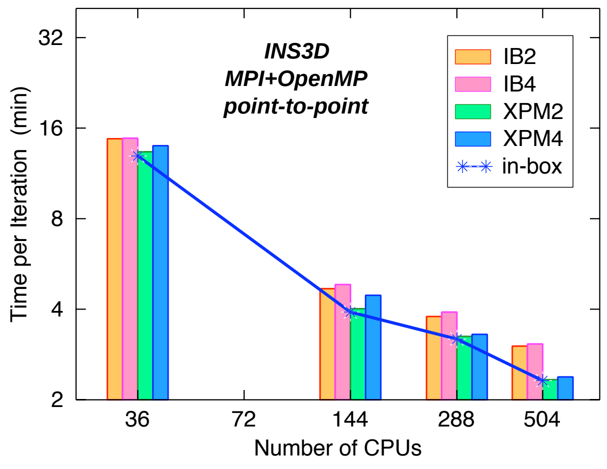

Figure 15. Performance of INS3D across multiple BX2b nodes via NUMAlink4 and InfinBand (MPI point-to-point communication).

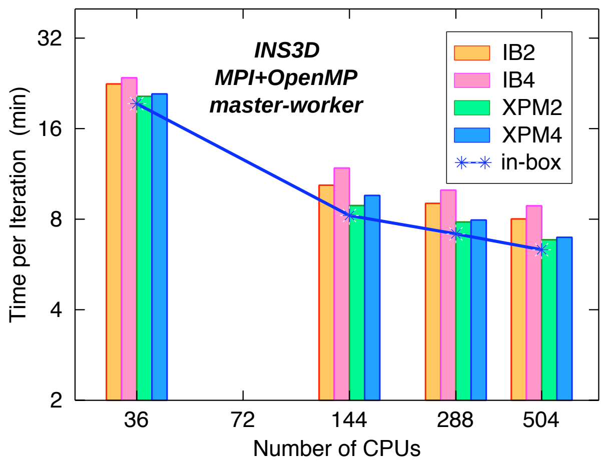

Figure 16. Performance of INS3D across multiple BX2b nodes via NUMAlink4 and InfinBand (MPI master-worker communication).

in runtime is observed when using InfiniBand instead of NUMAlink4; a similar increase was observed when using point-to-point communication as shown in Fig. 15.

4. *OVERFLOW-D*

Figure 17 displays OVERFLOW-D performance results obtained on multiple BX2b nodes. The total execution times are reported for the same number of processor counts via both NUMAlink4 and InfiniBand interconnects. The execution times for NUMAlink4 are generally 5–10% better; however, the reverse appears to be true for the communication times (not shown). Up until 508 CPUs, we did not observe a significant change in the execution timing for the same total number of processors distributed across multiple nodes via NUMAlink4 or InfiniBand, in comparison to the corresponding data obtained within a single node. The overall performance scalability is rather poor for the problem used in these experiments, and is adversely affected by the granularity of the grid blocks and increased overhead for large processor counts. In fact, as seen from Fig. 17, the execution times actually increase beyond 508 CPUs.

It should be noted that the shared I/O file system across multiple nodes that was available at the time of this study was much less efficient than the one used within a single node. Since the execution time includes the overhead for some minor I/O activities, albeit negligible for a single node, it is negatively affected to some extent for multiple nodes.

For the same total number of processors, the execution time for OVERFLOW-D across multiple nodes is less than the corresponding run on a single node (compare the stacked chart for multinode and the line plot for single node in Fig. 17). Our measurements indicate that communication times decrease significantly when using multiple boxes (and computation times actually increase somewhat). We speculate that this behavior may be due to the underlying send/receive buffer algorithm implemented in the MPI software used to handle large number of communications, such as all-to-all in this case, that perhaps benefits from the availability of more bandwidth in the multinode system. Bandwidth plays a more crucial role in the execution time than the latency for the communication pattern in our application. Note that this behavior is not

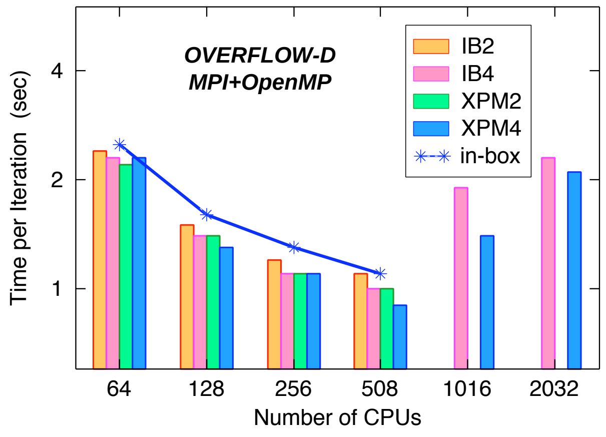

Figure 17. Performance of OVERFLOW-D across multiple BX2b nodes via NUMAlink4 and InfiniBand interconnects.

observed in the case of the INS3D application which is also based on overset grid methodology. We believe this is related to the communication patterns employed by OVERFLOW-D and INS3D, i.e. all-to-all versus point-to-point communication, respectively. In addition, the number of grid blocks used by INS3D for the experiments in this paper is much less than that used by OVERFLOW-D.

V. Summary and Conclusions

In this paper, we investigated the suitability of the Columbia supercomputer at NASA as a capability computing platform for aeronautics applications. Our benchmarking experiments demonstrated several features about single-box SGI Altix performance. First, the presence of the NUMAlink4 interconnect on the BX2 nodes provides a large performance boost for MPI and OpenMP applications. Furthermore, when the processor speed and cache size are enhanced (as is the case on those nodes we call BX2b's), there is another significant improvement in performance.

When multiple Altix nodes are combined into a capability cluster, both NUMALink4 and InfiniBand are capable of delivering very good performance. There are some caveats, however. For example, in the case of the master-worker version of INS3D, we observed that contention in the interconnect increased execution time substantially. On the other hand, the point-to-point version of INS3D scaled very well. Thus, careful attention should be paid to the choice of the communication strategy. With a suitable selection, we can scale some important applications to 2048 processors, as the work on Cart3D demonstrates.

In regard to scaling beyond that point, it is particularly encouraging that InfiniBand performed nearly as well as NUMAlink4 on the 2048-CPU shared-memory subcluster. For jobs using more than 2048 processors, InfiniBand is a necessity. However, because of the limitations of the InfiniBand hardware, doing so will require that a multilevel parallel programming paradigm be used. The excellent results from hybrid MPI+OpenMP and MPI+MLP benchmark runs are therefore very promising.

We also see great promise in the NAS Parallel Benchmark timings obtained on 4096-CPU subcluster. In the future, we will explore application scaling to that level. We will also investigate the causes of scalability problems that were observed with OpenMP, and experiment with the SGI SHMEM library.

Acknowledgments

We would like to thank Bron Nelson, Davin Chan, Bob Ciotti, and Bill Thigpen for their assistance in using Columbia, and Jeff Becker and Nateri Madavan for valuable comments on the manuscript. Rob Van der Wijngaart played a critical role in developing the multi-zone NPBs.

References

[1]M. J. Aftosmis, M. J. Berger, and G. D. Adomavicius. A parallel multilevel method for adaptively refined Cartesian grids with embedded boundaries. In *Proc. 38th AIAA Aerospace Sciences Meeting & Exhibit*, Paper 2000-0808, Reno, NV, Jan. 2000.

[2]M. J. Aftosmis, M. J. Berger, and S. M. Murman. Applications of space-filling-curves to Cartesian methods in CFD. In *Proc. 42nd AIAA Aerospace Sciences Meeting & Exhibit*, Paper 2004-1232, Reno, NV, Jan. 2004.

[3]D. Bailey, J. Barton, T. Lasinski, and H. Simon (Eds.). The NAS Parallel Benchmarks. Technical Report NAS-91-002, NASA Ames Research Center, Moffett Field, CA, 1991.

[4]D. Bailey, T. Harris, W. Saphir, R. Van der Wijngaart, A. Woo, and M. Yarrow. The NAS Parallel Benchmarks 2.0. Technical Report NAS-95-020, NASA Ames Research Center, Moffett Field, CA, 1995.

[5]M. J. Berger, M. J. Aftosmis, D. D. Marshall, and S. M. Murman. Performance of a new CFD flow solver using a hybrid programming paradigm. *Journal of Parallel and Distributed Computing*, 65:414–423, 2005.

[6]R. Biswas, M. J. Djomehri, R. Hood, H. Jin, C. Kiris, and S. Saini. An applications-based performance characterization of the Columbia supercluster. In *Proc. SC|05*, Seattle, WA, Nov. 2005.

[7]P. G. Buning, D. C. Jespersen, T. H. Pulliam, W. M. Chan, J. P. Slotnick, S. E. Krist, and K. J. Renze. Overflow user's manual, version 1.8g. Technical report, NASA Langley Research Center, Hampton, VA, 1999.

[8]M. J. Djomehri and R. Biswas. Performance analysis of a hybrid overset multi-block application on multiple architectures. In *Proc. 10th International Conference on High Performance Computing (HiPC)*, pages 383–392, Hyderabad, India, Dec. 2003.

[9]M. J. Djomehri, R. Biswas, and N. Lopez-Benitez. Load balancing strategies for multi-block overset grid applications. In *Proc. 18th International Conference on Computers and Their Applications (CATA)*, pages 373–378, Honolulu, HI, Mar. 2003.

[10]InfiniBand Specifications. http://www.infinibandta.org/specs.

[11]H. Jin and R. Van der Wijngaart. Performance characteristics of the multi-zone NAS Parallel Benchmarks. In *Proc. International Parallel and Distributed Processing Symposium (IPDPS)*, Santa Fe, NM, Apr. 2004.

[12]C. Kiris, D. Kwak, and W. M. Chan. Parallel unsteady turbopump simulations for liquid rocket engines. In *Proc. SC2000*, Dallas, TX, Nov. 2000.

[13]C. Kiris, D. Kwak, and S. Rogers. Incompressible Navier-Stokes solvers in primitive variables and their applications to steady and unsteady flow simulations. In M. Hafez, editor, *Numerical Simulations of Incompressible Flows*. World Scientific, 2003.

[14]J. Liu, B. Chandrasekaran, J. Wu, W. Jiang, S. Kini, W. Yu, D. Buntinas, P. Wyckoff, and D. Panda. Performance comparison of MPI implementations over InifiniBand, Myrinet, and Quadrics. In *Proc. SC2003*, Phoenix, AZ, Nov. 2003.

[15]D. J. Mavriplis, M. J. Aftosmis, and M. J. Berger. High resolution aerospace applications using the NASA Columbia supercomputer. In *Proc. SC|05*, Seattle, WA, Nov. 2005.

[16]R. Meakin and A. M. Wissink. Unsteady aerodynamic simulation of static and moving bodies using scalable computers. In *Proc. 14th AIAA Computational Fluid Dynamics Conference*, Paper 99-3302, Norfolk, VA, June 1999.

[17]S. M. Murman, M. J. Aftosmis, and M. Nemec. Automated parameter studies using a Cartesian method. In *Proc. 22nd AIAA Applied Aerodynamics Conference & Exhnibit*, Paper 2004-5076, Providence, RI, June 2004.

[18]NAS Parallel Benchmarks. http://www.nas.nasa.gov/Software/NPB.

[19]R. C. Strawn and M. J. Djomehri. Computational modeling of hovering rotor and wake aerodynamics. *Journal of Aircraft*, 39(5):786–793, 2002.

[20]J. R. Taft. Achieving 60 Gflop/s on the production CFD code OVERFLOW-MLP. *Parallel Computing*, 27(4):521–536, 2001.

[21]Top500 Supercomputer Sites. http://www.top500.org.

[22]Voltaire ISR 9288 InfiniBand switch router. http://www.voltaire.com/documents/9288dsweb.pdf.